

Forouzan

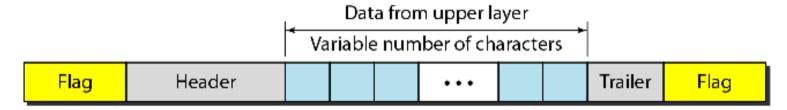
Chapter 11 Data Link Control

Data Link Layer

- The two main functions of data link layer are
 - Data link control
 - 2. Media access
- Data link control includes
 - Framing
 - 2. Flow and error control
 - 3. And protocols

11-1 FRAMING

The data link layer needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.



Topics discussed in this section:

- 1. Fixed-Size Framing
- 2. Variable-Size Framing
 - 1. Character oriented approach
 - 2. Bit oriented approach

11-1 FRAMING

1. Character-oriented protocol:

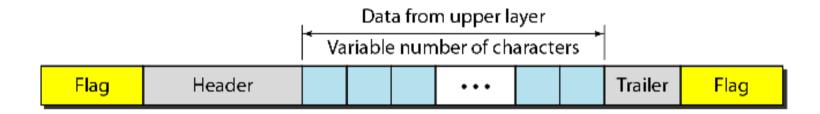
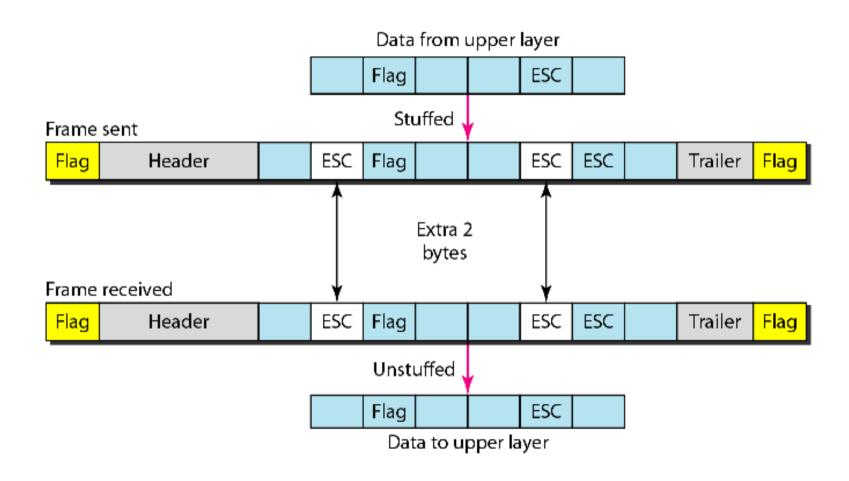


Figure 11.1 A frame in a character-oriented protocol

Figure 11.2 Byte stuffing and unstuffing





Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

2. bit-oriented protocol: uses a special bit pattern 011111110

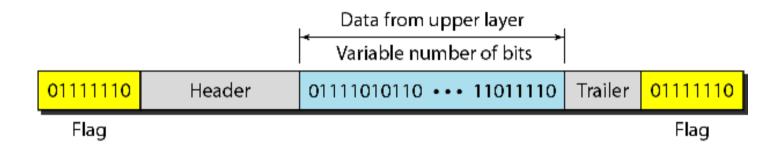
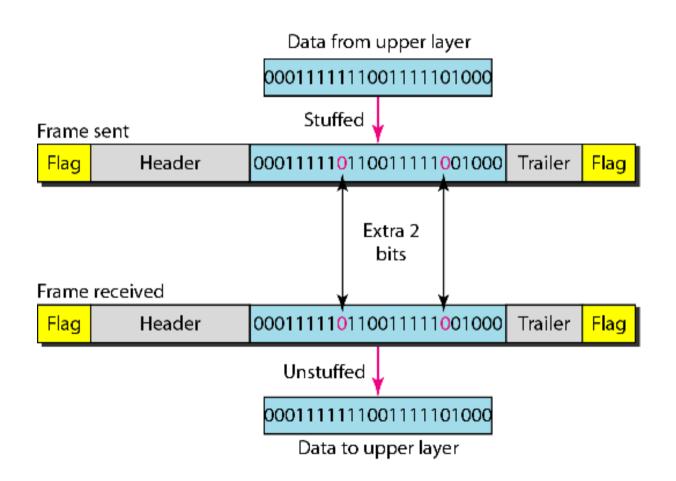


Figure 11.3 A frame in a bit-oriented protocol



Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.

Figure 11.4 Bit stuffing and unstuffing



11-2 FLOW AND ERROR CONTROL

The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

Topics discussed in this section:

Flow Control : Coordinates the amount of data that can be sent.

Error Control: Error detection and Correction



Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.

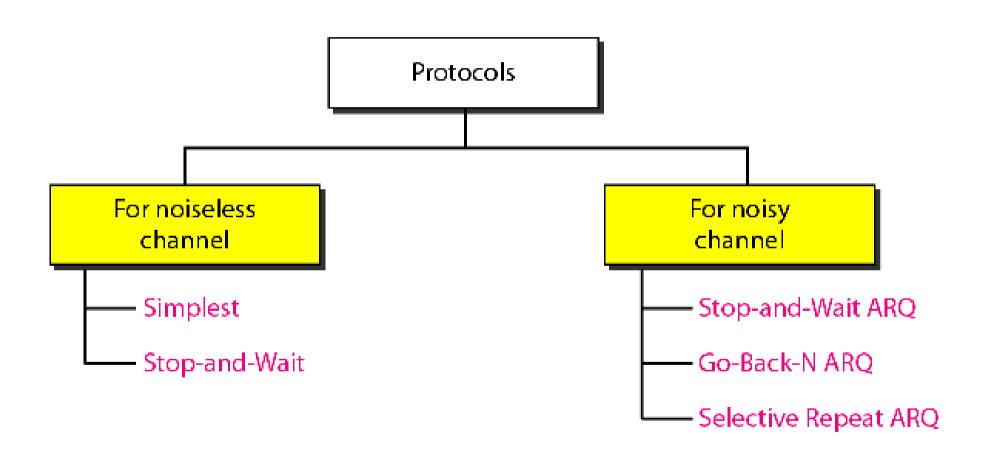


Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

11-3 PROTOCOLS

Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages. To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.

Figure 11.5 Taxonomy of protocols discussed in this chapter



11-4 NOISELESS CHANNELS

Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.

Topics discussed in this section:

Simplest Protocol Stop-and-Wait Protocol

Simplest Protocol

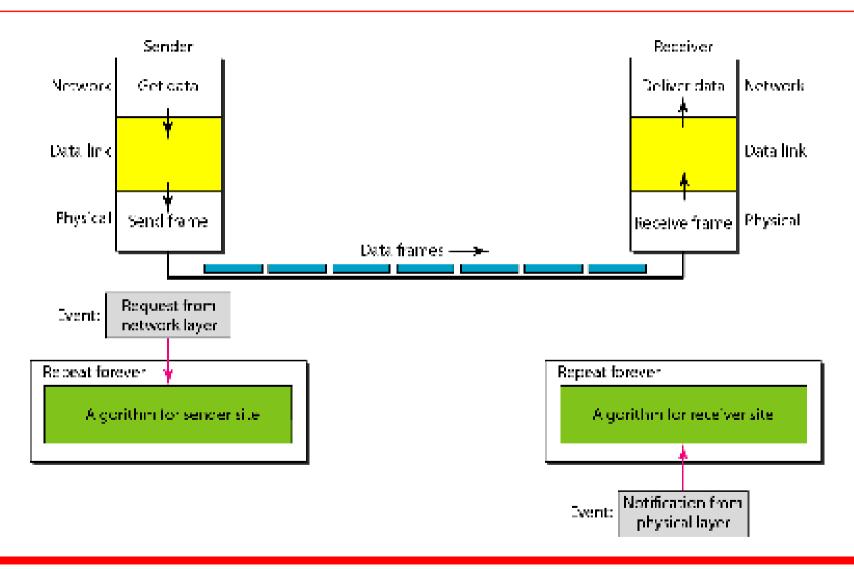


Figure 11.6 The design of the simplest protocol with no flow or error control

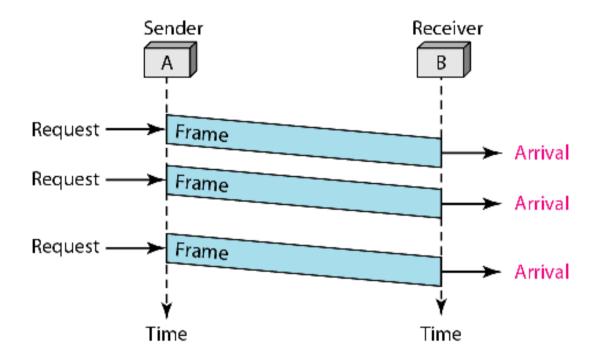
Algorithm 11.1 Sender-site algorithm for the simplest protocol

Algorithm 11.2 Receiver-site algorithm for the simplest protocol

Example 11.1

Figure 11.7 shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.

Figure 11.7 Flow diagram for Example 11.1



Stop and wait Protocol

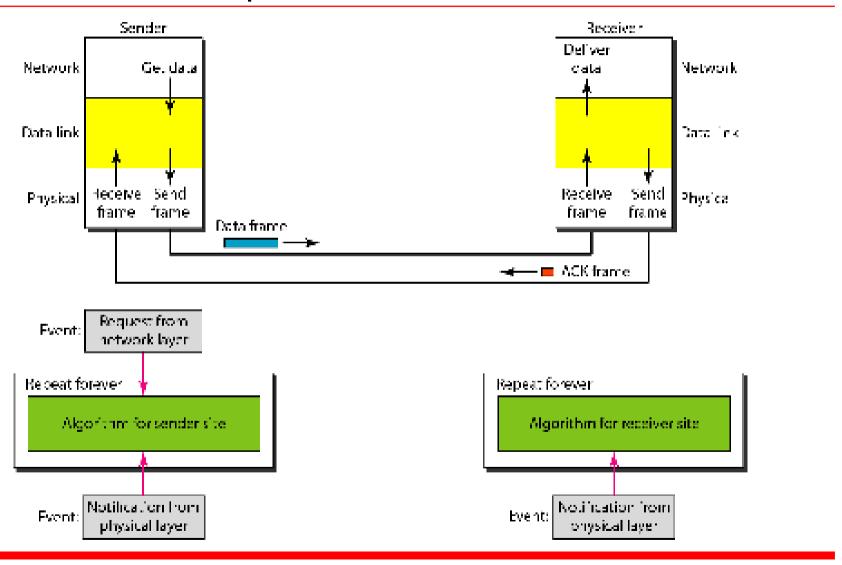


Figure 11.8 Design of Stop-and-Wait Protocol

Algorithm 11.3 Sender-site algorithm for Stop-and-Wait Protocol

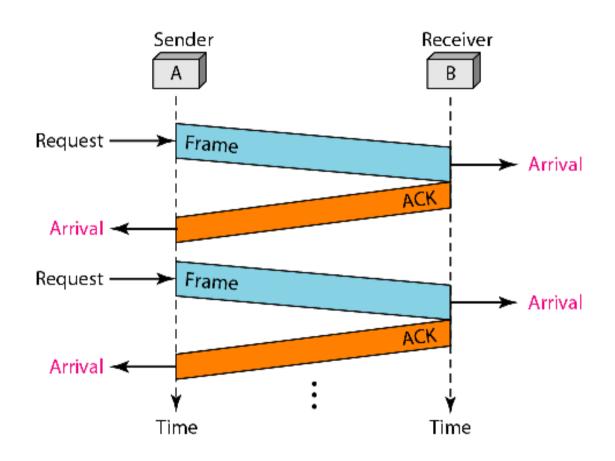
```
while(true)
                               //Repeat forever
  canSend = true
                               //Allow the first frame to go
3
4
    WaitForEvent(); // Sleep until an event occurs
    if (Event (RequestToSend) AND canSend)
 6
       GetData();
      MakeFrame():
       SendFrame();
                   //Send the data frame
       canSend = false;  //Cannot send until ACK arrives
10
11
12
    WaitForEvent(); // Sleep until an event occurs
    if (Event (ArrivalNotification) // An ACK has arrived
13
14
15
       ReceiveFrame(); //Receive the ACK frame
16
       canSend = true:
17
18
```

Algorithm 11.4 Receiver-site algorithm for Stop-and-Wait Protocol

Example 11.2

Figure 11.9 shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

Figure 11.9 Flow diagram for Example 11.2



11-5 NOISY CHANNELS

Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.

Topics discussed in this section:

Stop-and-Wait Automatic Repeat Request Go-Back-N Automatic Repeat Request Selective Repeat Automatic Repeat Request

Stop-and-Wait Automatic Repeat Request.

Issues

- 1. How does receiver detect corrupted frames?
- 2. How does sender know about the errors or lost frames?
- 3. How does receiver know about lost or duplicate frames?
- 4 How does receiver know about lost ACK?

Stop-and-Wait Automatic Repeat Request.

- Sequence Numbers:
 - A field is added to the data frame to hold the sequence number of that frame.
 - The sequence numbers of course can wrap around.
 - The smallest range is 2 (x and x+1)
 - It can be x=0 and x+1 = 1;
- Acknowledge numbers:
 - A field is added to the ACK frame to hold the sequence number of that ACK frame.



Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.



In Stop-and-Wait ARQ, we use sequence numbers to number the frames.

The sequence numbers are based on modulo-2 arithmetic.



In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Stop-and-Wait Automatic Repeat Request.

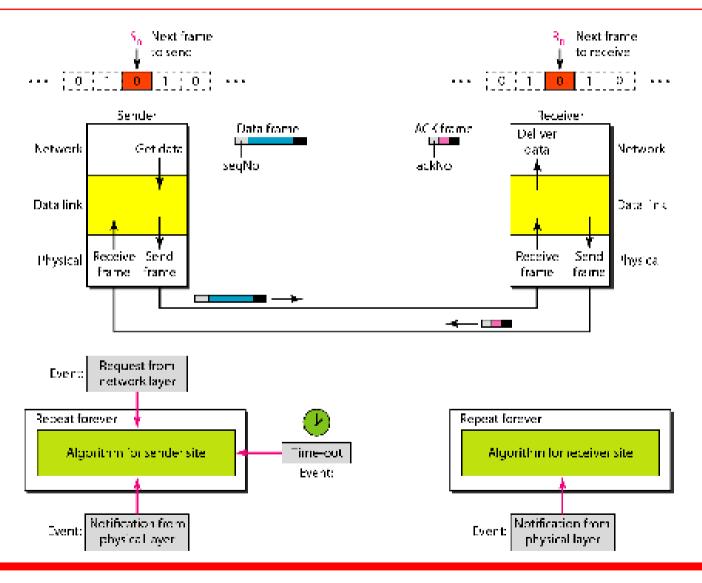


Figure 11.10 Design of the Stop-and-Wait ARQ Protocol

11.31

Algorithm 11.5 Sender-site algorithm for Stop-and-Wait ARQ

```
S_n = 0;
                                 // Frame 0 should be sent first
 2 canSend = true;
                                 // Allow the first request to go
  while(true)
                                 // Repeat forever
     WaitForEvent():
                                // Sleep until an event occurs
     if(Event(RequestToSend) AND canSend)
        GetData():
        MakeFrame(S_n);
                                          //The seqNo is S_n
        StoreFrame(S_n);
10
                                          //Keep copy
11
        SendFrame(S_n);
12
        StartTimer();
13
        S_n = S_n + 1;
        canSend = false;
14
15
16
     WaitForEvent();
                                           // Sleep
```

(continued)

Algorithm 11.5 Sender-site algorithm for Stop-and-Wait ARQ(continued)

```
if(Event(ArrivalNotification) // An ACK has arrived
17
18
19
         ReceiveFrame(ackNo): //Receive the ACK frame
         if(not corrupted AND ackNo == Sn) //Valid ACK
20
21
22
             Stoptimer();
23
             PurgeFrame (S_{n-1});
                                       //Copy is not needed
24
             canSend = true;
25
26
27
28
       if(Event(TimeOut)
                                          // The timer expired
29
30
        StartTimer():
31
        ResendFrame(S_{n-1});
                                        //Resend a copy check
32
33
```

Algorithm 11.6 Receiver-site algorithm for Stop-and-Wait ARQ Protocol

```
// Frame 0 expected to arrive first
   R_n = 0;
  while(true)
 3
    WaitForEvent(); // Sleep until an event occurs
     if(Event(ArrivalNotification)) //Data frame arrives
        ReceiveFrame():
        if(corrupted(frame));
           sleep();
10
        if(seqNo == R_n)
                                     //Valid data frame
11
12
         ExtractData();
13
                                      //Deliver data
          DeliverData():
14
          R_n = R_n + 1;
15
16
         SendFrame (R_n);
                                     //Send an ACK
17
18
```



Figure 11.11 shows an example of Stop-and-Wait ARQ. Frame 0 is sent and acknowledged. Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops. Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.

Stop-and-Wait Automatic Repeat Request. Request **C** 1 0; 1;0;1 ⊫[0 🚺 0 1 0 1 Arrival Stop Arrival $C = C(1)O(1 \rightarrow$ 5,, Request 0 1 0;1;0;1 Frame 1 Time-out | 0 | 1 | 0 | 1 | 0 | 1 restart Arriva (0:1 0 0 1 → Stop Request (0) 1 0 1 0 1 ⊷(o n o<mark>n </mark>ojn Arrival Time-out Time-out {0{ 1 0 1 restart Discard, dup idate Arriva (0,1(0 0 1)⊸ Stop

Figure 11.11 Flow diagram for Example 11.3

Time

Stop-and-Wait Automatic Repeat Request.

Efficiency:

If the bandwidth of the channel is large then this protocol becomes inefficiency.

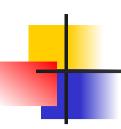
Example 11.4

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

 $(1 \times 10^6) \times (20 \times 10^{-3}) = 20,000 \text{ bits}$



Example 11.4 (continued)

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.



What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits. The system can send up to 15 frames or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

Go-Back-N Protocol

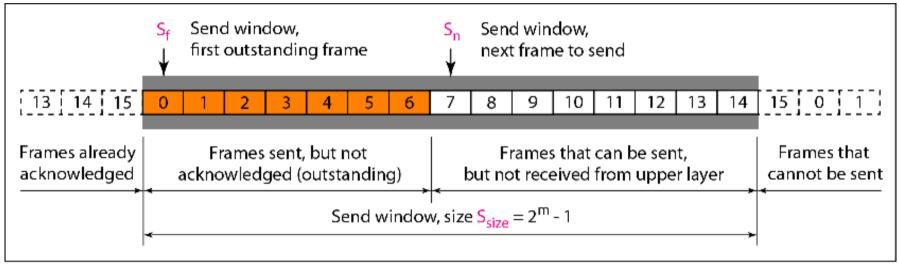
- Issues
 - Sequence numbers
 - The size is 2^m 1
 Example: m=4
 0,1,2...14,15,0,1,2..14,15,0,1...
- Sliding window
 - Send sliding window
 - Receive sliding window



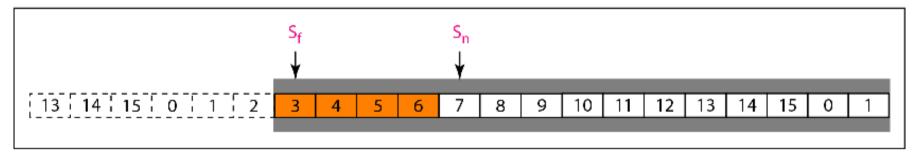
Note

In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Figure 11.12 Send window for Go-Back-N ARQ



a. Send window before sliding



b. Send window after sliding



Note

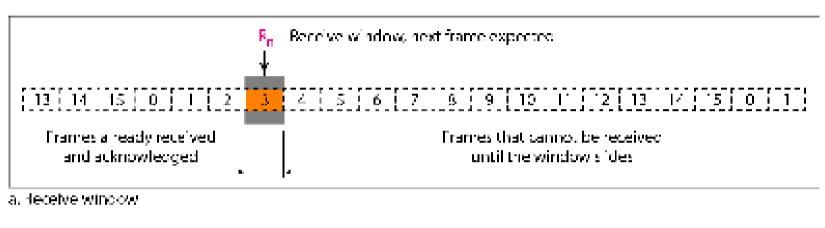
The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f , S_n , and S_{size} .

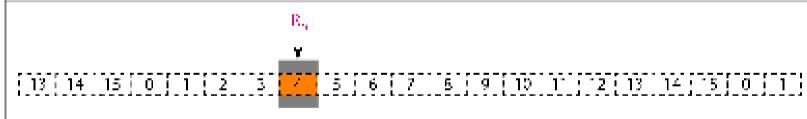


Note

The send window can slide one or more slots when a valid acknowledgment arrives.

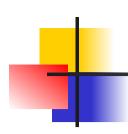
Receiver window





b. Window after sliding

Figure 11.13 Receive window for Go-Back-N ARQ



Note

The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n.

The window slides when a correct frame has arrived; sliding occurs one slot at a time.

Figure 11.14 Design of Go-Back-N ARQ

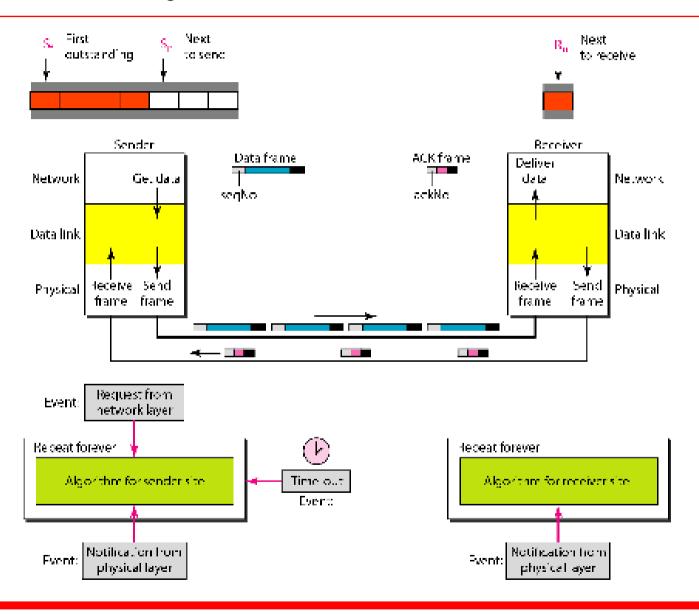
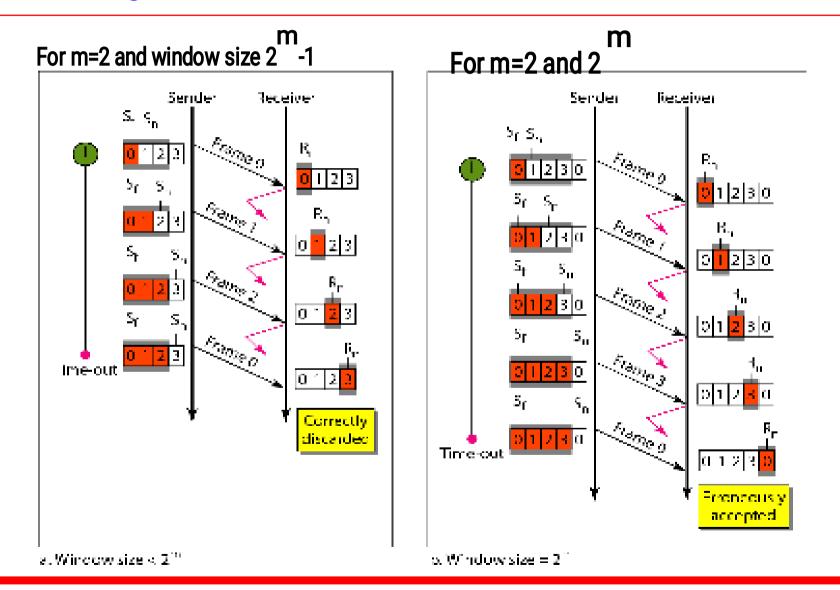


Figure 11.15 Window size for Go-Back-N ARQ





Note

In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Algorithm 11.7 Go-Back-N sender algorithm

```
S_w = 2^m - 1;
 2 S_f = 0;
 3 S_n = 0;
   while (true)
                                         //Repeat forever
 6
    WaitForEvent();
     if(Event(RequestToSend))
                                        //A packet to send
 9
                                        //If window is full
10
         if(S_n-S_f >= S_w)
               Sleep();
11
12
         GetData();
13
         MakeFrame(S_n);
14
         StoreFrame (S_n);
15
         SendFrame (S_n);
         S_n = S_n + 1;
16
         if(timer not running)
17
              StartTimer();
18
19
20
```

(continued)

Algorithm 11.7 Go-Back-N sender algorithm

(continued)

```
21
      if(Event(ArrivalNotification)) //ACK arrives
22
23
         Receive (ACK):
         if(corrupted(ACK))
24
25
               Sleep();
26
         if((ackNo>S_f)&&(ackNo<=S_n)) //If a valid ACK
         While (S_f \le ackNo)
27
28
29
           PurgeFrame(S<sub>f</sub>);
30
           S_f = S_f + 1;
31
32
          StopTimer();
33
34
35
      if(Event(TimeOut))
                                           //The timer expires
36
       StartTimer();
37
38
       Temp = S_{f};
39
       while (Temp < S_n);
40
         SendFrame(S<sub>f</sub>);
41
42
         S_f = S_f + 1;
43
44
4.5
```

Algorithm 11.8 Go-Back-N receiver algorithm

```
\mathbf{R}_{\mathbf{n}} = \mathbf{0};
   while (true)
                                          //Repeat forever
 4
     WaitForEvent():
 6
      if (Event (ArrivalNotification)) / Data frame arrives
         Receive (Frame);
         if(corrupted(Frame))
10
               Sleep();
11
                                          //If expected frame
12
         if(seqNo == R_n)
13
14
           DeliverData():
                                        //Deliver data
15
           R_n = R_n + 1;
                                          //Slide window
16
           SendACK(R_n);
17
18
      }
19
```



Figure 11.16 shows an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost. After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer. There is no time-out event here because all outstanding frames are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 11.54 ACK 3.

Figure 11.16 Flow diagram for Example 11.6

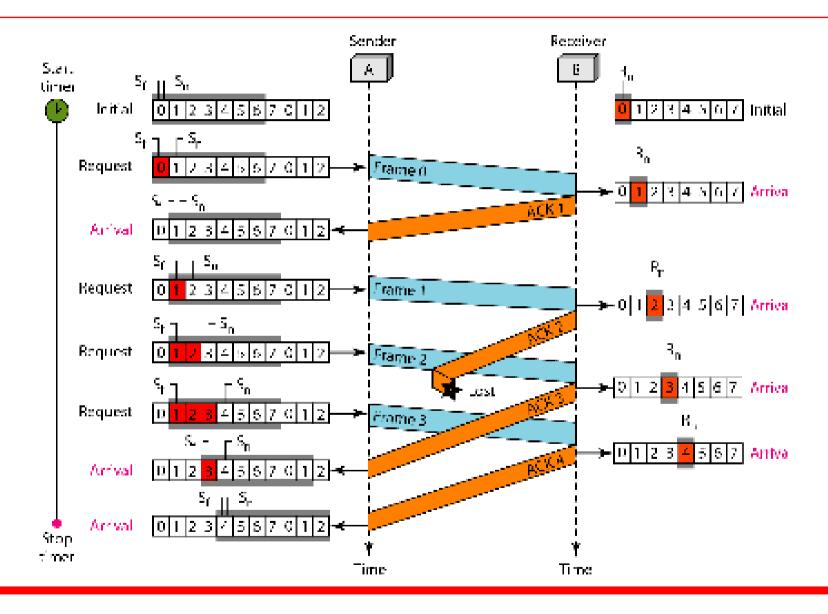




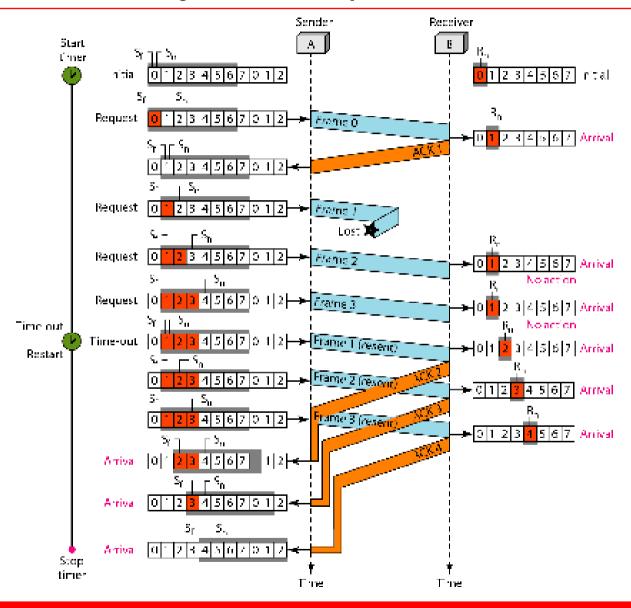
Figure 11.17 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.



Example 11.7 (continued)

The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.

Figure 11.17 Flow diagram for Example 11.7





Note

Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

Figure 11.18 Send window for Selective Repeat ARQ

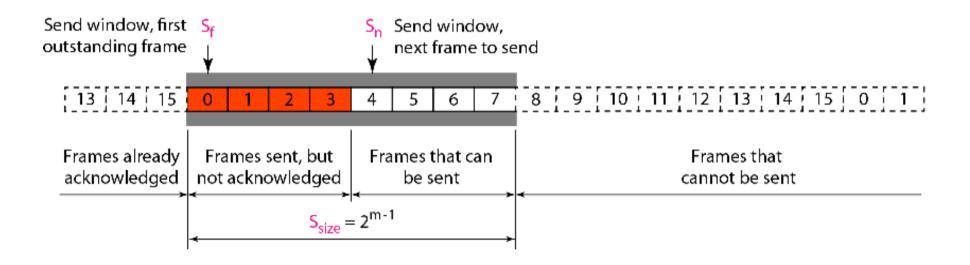


Figure 11.19 Receive window for Selective Repeat ARQ

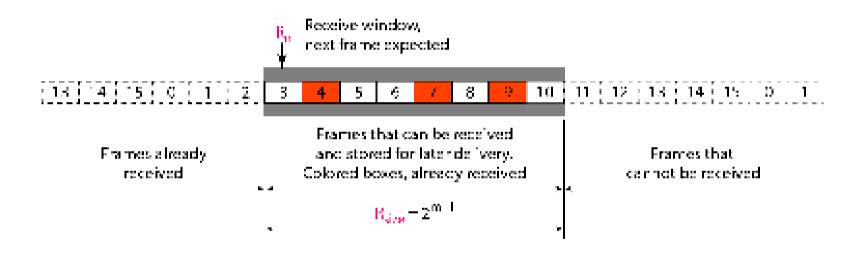


Figure 11.20 Design of Selective Repeat ARQ

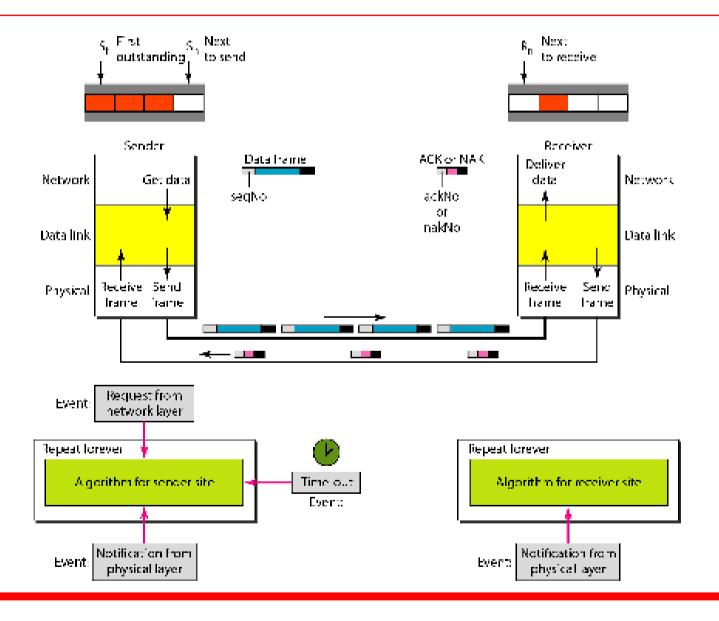
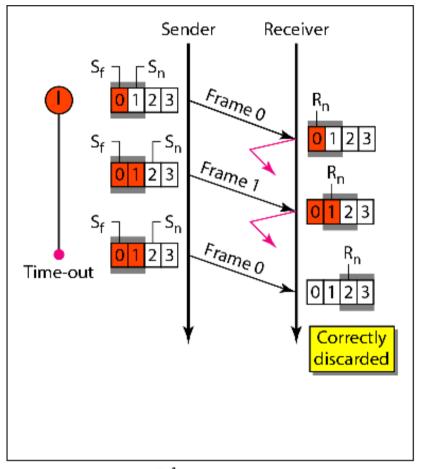
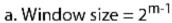
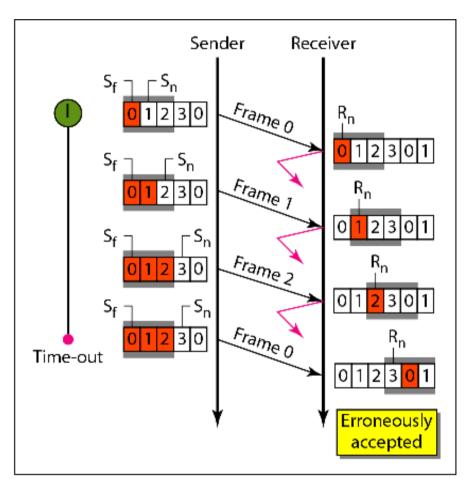


Figure 11.21 Selective Repeat ARQ, window size







b. Window size $> 2^{m-1}$



Note

In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

Algorithm 11.9 Sender-site Selective Repeat algorithm

```
S_w = 2^{m-1} ;
   S_f = 0;
 4
   while (true)
                                         //Repeat forever
 6
     WaitForEvent():
     if(Event(RequestToSend))
                                         //There is a packet to send
 9
10
         if(S_n-S_f >= S_w)
                                        //If window is full
11
               Sleep();
12
         GetData();
13
         MakeFrame(S_n);
14
         StoreFrame(S_n);
15
         SendFrame (S_n);
         S_n = S_n + 1;
16
17
         StartTimer(S_n);
18
19
```

(continued)

Algorithm 11.9 Sender-site Selective Repeat algorithm (continued)

```
if(Event(ArrivalNotification)) //ACK arrives
20
21
22
         Receive(frame):
                                          //Receive ACK or NAK
23
         if(corrupted(frame))
24
               Sleep();
25
         if (FrameType == NAK)
26
             if (nakNo between S_f and S_n)
27
28
              resend(nakNo);
29
              StartTimer(nakNo);
30
31
         if (FrameType == ACK)
32
             if (ackNo between S_f and S_n)
33
34
               while (s_f < ackNo)
35
36
                Purge(s<sub>f</sub>);
37
                StopTimer(s<sub>f</sub>);
                S_f = S_f + 1;
38
39
40
41
```

Algorithm 11.9 Sender-site Selective Repeat algorithm

(continued)

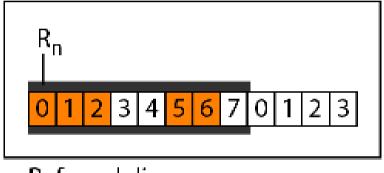
Algorithm 11.10 Receiver-site Selective Repeat algorithm

```
1 R_n = 0;
 2 NakSent = false;
 3 AckNeeded = false;
   Repeat (for all slots)
 5
        Marked(slot) = false;
 6
   while (true)
                                                 //Repeat forever
 8
 9
     WaitForEvent();
10
                                                 /Data frame arrives
11
     if(Event(ArrivalNotification))
12
13
         Receive (Frame);
14
         if(corrupted(Frame))&& (NOT NakSent)
15
16
          SendNAK(R<sub>n</sub>);
17
          NakSent = true;
18
          Sleep();
19
20
         if(seqNo <> Rn)&& (NOT NakSent)
21
22
          SendNAK(R<sub>n</sub>);
```

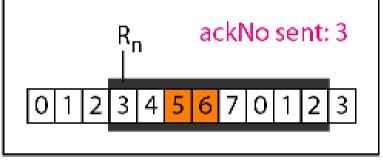
Algorithm 11.10 Receiver-site Selective Repeat algorithm

```
23
           NakSent = true;
24
           if ((seqNo in window)&&(!Marked(seqNo))
25
26
            StoreFrame (segNo)
            Marked(seqNo) = true;
27
28
            while (Marked (R<sub>n</sub>))
29
30
             DeliverData(R<sub>n</sub>);
31
             Purge(R<sub>n</sub>);
32
             R_n = R_n + 1;
             AckNeeded = true:
33
34
35
             if (AckNeeded);
36
37
             SendAck(R_n);
38
             AckNeeded = false;
             NakSent = false;
39
40
41
42
43
44
```

Figure 11.22 Delivery of data in Selective Repeat ARQ



a. Before delivery



b. After delivery



This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 shows the situation. One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives. The other two timers start when the corresponding frames are sent and stop at the last arrival event.



Example 11.8 (continued)

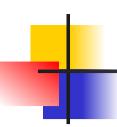
At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer. At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer. There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window.

11.72



Example 11.8 (continued)

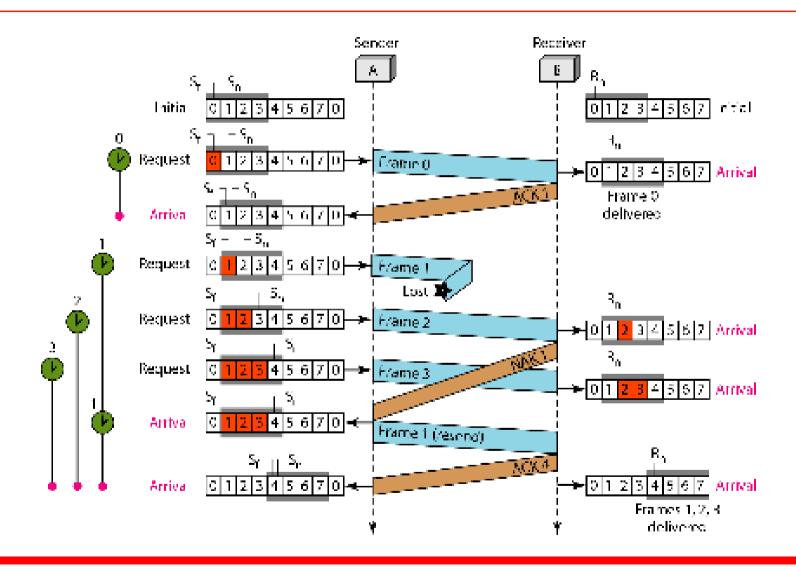
Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.



Example 11.8 (continued)

The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.

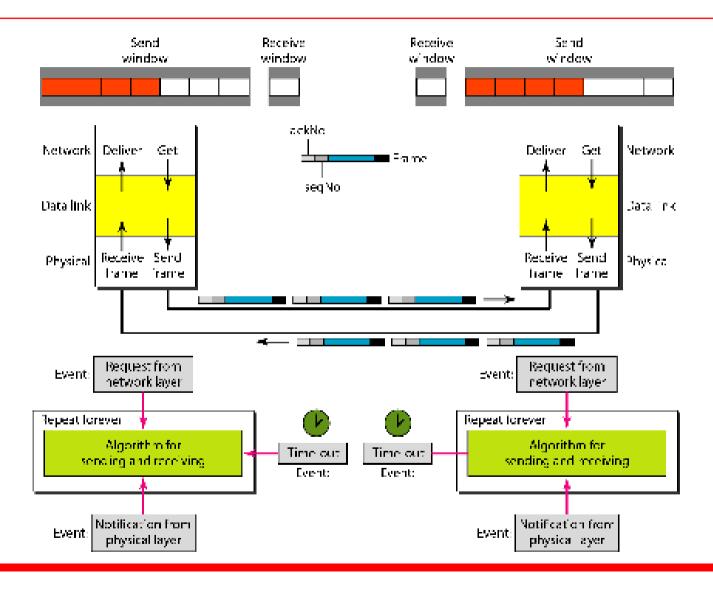
Figure 11.23 Flow diagram for Example 11.8



piggybacking

 A technique for improving the efficiency of the bidirectional protocols

Figure 11.24 Design of piggybacking in Go-Back-N ARQ



11-6 HDLC

High-level Data Link Control (HDLC) is a bitoriented protocol for communication over point-topoint and multipoint links. It implements the ARQ mechanisms we discussed in this chapter.

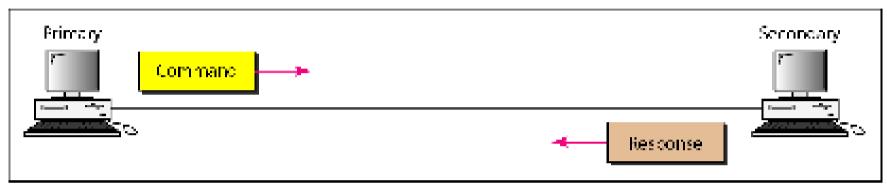
Topics discussed in this section:

Configurations and Transfer Modes Frames
Control Field

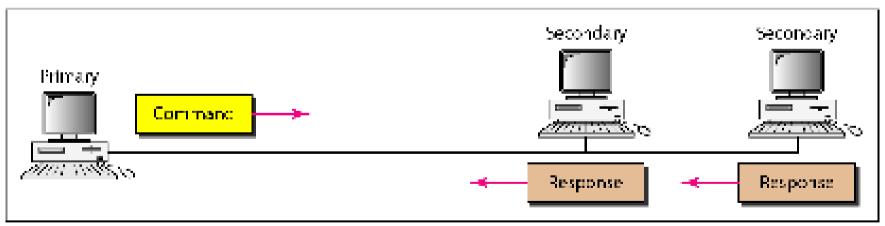
HDLC

- Configurations
 - Point to Pont link
 - Multipoint link
- Transfer mode
 - Normal response mode
 - Asynchronous balanced mode

Normal response mode



a. Point-te-point



b. Multipoint

Figure 11.25 Normal response mode

Asynchronous balanced mode

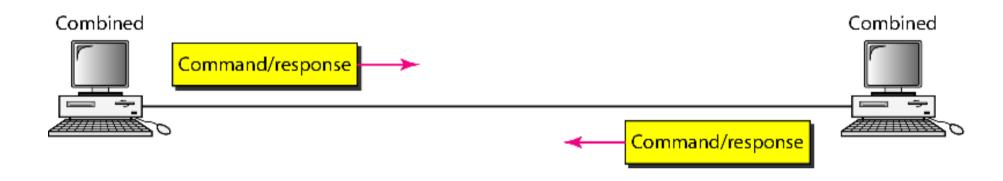


Figure 11.26 Asynchronous balanced mode

HDLC frames

- 1. Information frames (I-frames)
- 2. Supervisory Frames (S-frames)
- 3. Unnumbered Frames (U-frames)

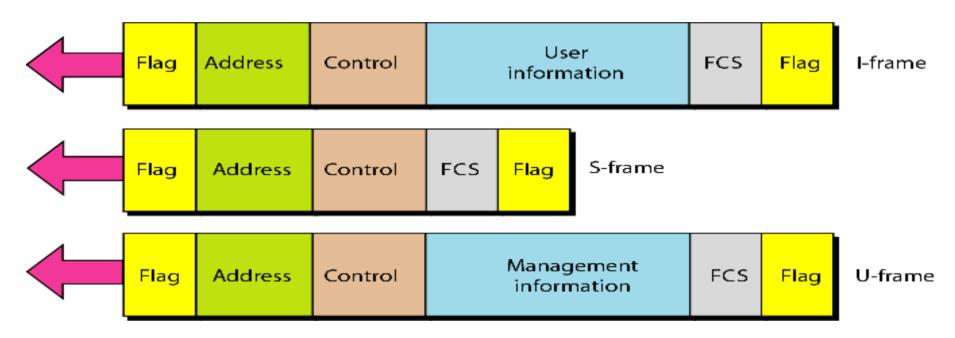


Figure 11.27 HDLC frames

HDLC frames

1. Fields

- 1. Flag fields: 8bit sequence (01111110)
- 2. Address field: 1byte or several bytes
- Control field: 1 or 2 byte (flow and error control)
- 4. Information field or management information : User data or system info.
- Frame cheque sequence (FCS): HDLC error detection filed(2 or 4 bytes).

The control field determines the type of frame and defines its functionality.

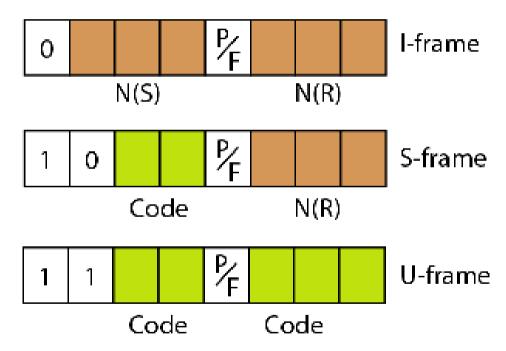


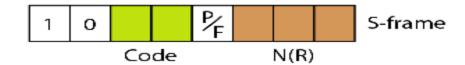
Figure 11.28 Control field format for the different frame types

control field for I frames



- First bit defines type
- •Next 3 bit N(S) define sequence number
- ·Last 3 bits N(S) defines acknowledgement number
- •P/F bit if set =1 can mean poll or final

control field for S- frames



In S-frames control fields

- •The first 2 bits, 10 means S-frames
- ·Last 3 bit ACK number or NACK
- •P/F bit has similar meaning
- •The 2 bit code defines 4 types of frames
- Receive Ready (RR -00)(acknoledges the safe and sound frame)
- •Receive Not Ready (RNR-10)(Reciever is busy and cannot receive)
- •Reject (REJ- 01)(NAK used for go back N protocol)
- **11.86** •Selective Reject (SREJ 11)(NAK for selective repeat)

control field for U- frames

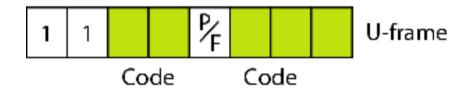


Table 11.1 U-frame control command and response

Code	Command	Response	Meaning
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	DM	Set asynchronous balanced mode or disconnect mode
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	UI	Unnumbered information
00-110		UA	Unnumbered acknowledgment
00 010	DISC	RD	Disconnect or request disconnect
10 000	SIM	RIM	Set initialization mode or request information mode
00-100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	XID	Exchange ID
10 001	FRMR	FRMR	Frame reject.



Figure 11.29 shows how U-frames can be used for connection establishment and connection release. Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment).

Figure 11.29 Example of connection and disconnection

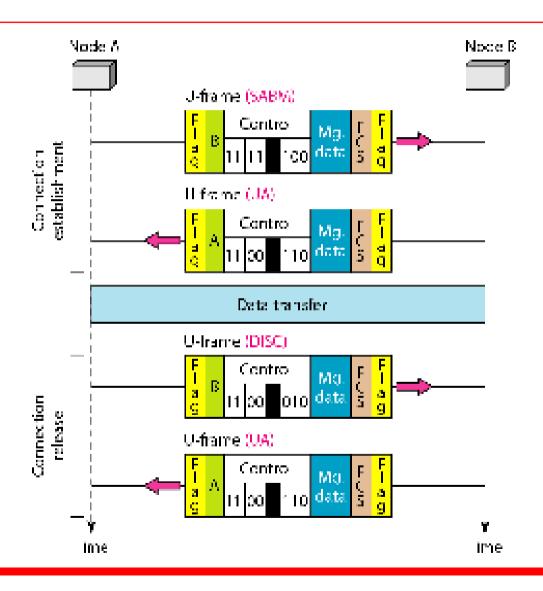
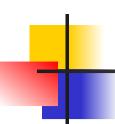




Figure 11.30 shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1. Node B piggybacks its acknowledgment of both frames onto an I-frame of its own. Node B's first

I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of A's frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A.



Example 11.10 (continued)

Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting A's frame 2 to arrive next. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. The RR code indicates that A is still ready to receive. The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3.

Figure 11.30 Example of piggybacking without error

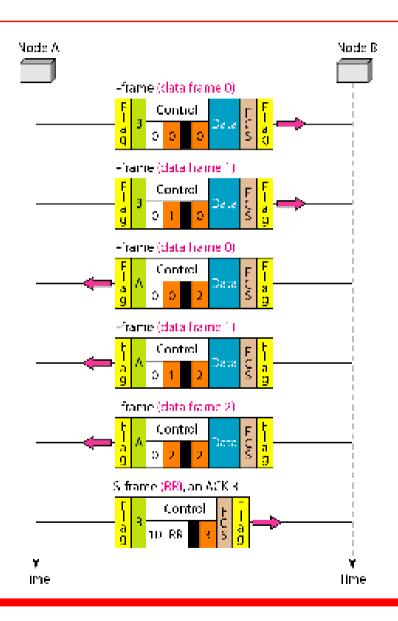
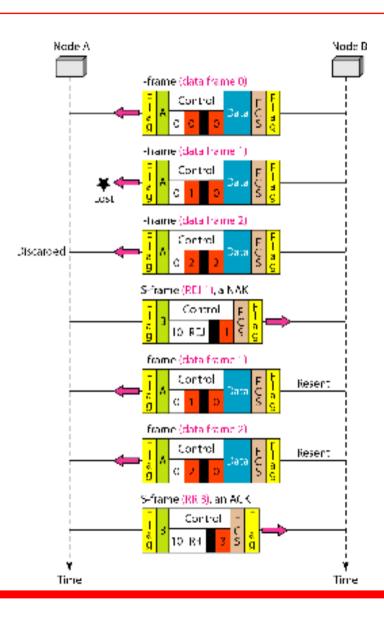




Figure 11.31 shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost. When node A receives frame 2, it discards it and sends a REJ frame for frame 1. Note that the protocol being used is Go-Back-N with the special use of an REJ frame as a NAK frame. The NAK frame does two things here: It confirms the receipt of frame 0 and declares that frame 1 and any following frames must be resent. Node B, after receiving the REJ frame, resends frames 1 and 2. Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.

Figure 11.31 Example of piggybacking with error



11-7 POINT-TO-POINT PROTOCOL

Although HDLC is a general protocol that can be used for both point-to-point and multipoint configurations, one of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). PPP is a byte-oriented protocol.

Topics discussed in this section:

Framing
Transition Phases
Multiplexing
Multilink PPP

11-7 POINT-TO-POINT PROTOCOL

- PPP provides several services
 - PPP defines the format of the frame to be exchanged.
 - PPP defines how two devices can negotiate the establishment of the link and the exchange of data
 - PPP defines how network layer data are encapsulated in the data link frame
 - PPP defines two services can authenticate each other
 - PPP provides multiple network layer services supporting variety of network layer protocols
 - PPP provides connection over multiple links
 - PPP provides network address configuration.

11-7 POINT-TO-POINT PROTOCOL

- What is missing in PPP?
 - PPP does not provide flow control.
 - PPP has a very simple mechanism for error control.
 - PPP does not provide a sophisticated addressing mechanism to handle frames in a multiple point configuration.

PPP frame format

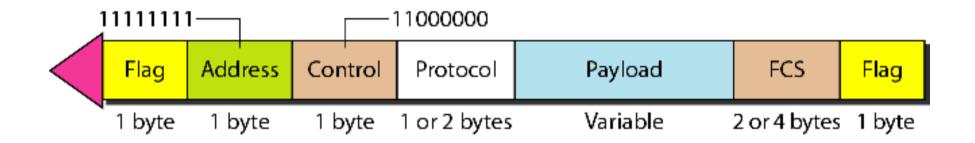


Figure 11.32 PPP frame format



Note

PPP is a byte-oriented protocol using byte stuffing with the escape byte 01111101.

PPP Transition phases

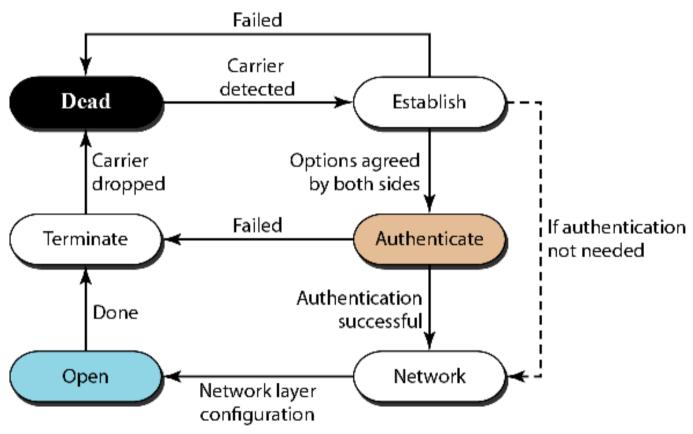


Figure 11.33 Transition phases

Multiplexing in PPP

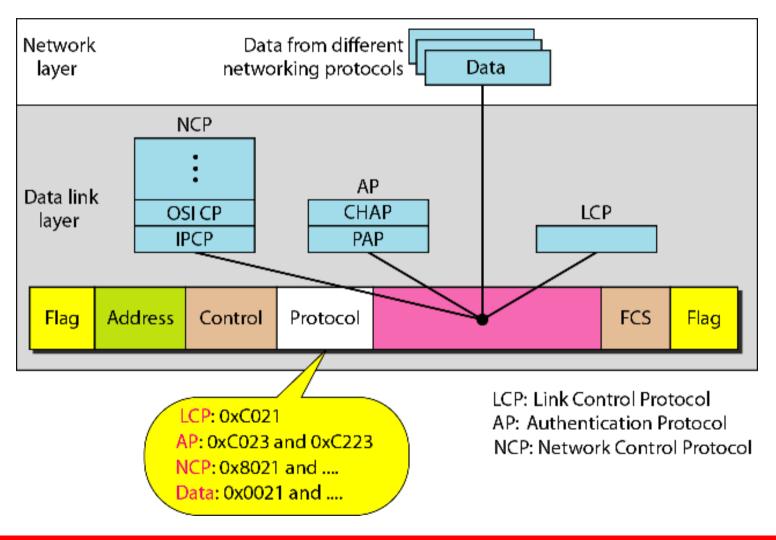


Figure 11.34 Multiplexing in PPP

Multiplexing in PPP

Link control protocol:

- Responsible for establishing maintaining ,configuring and terminating links.
- •Provides negotiation mechanism to set options between the two end points.
- The protocol field is set to C021
- ·It supports 11 types of packets

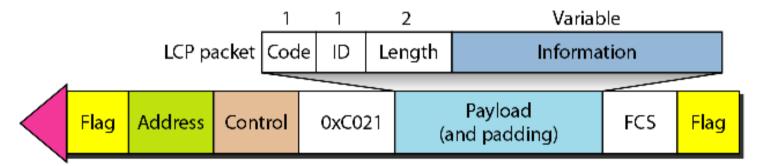


Figure 11.35 LCP packet encapsulated in a frame

Table 11.2 LCP packets

Code	Packet Type	Description
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

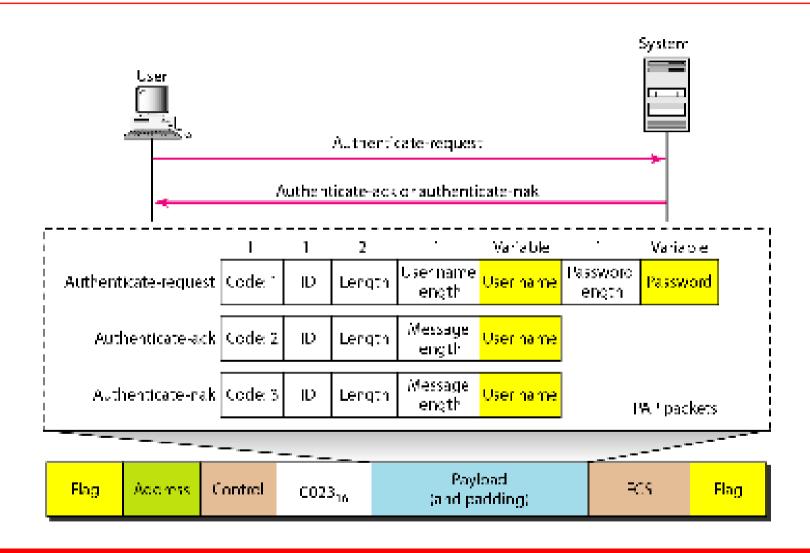
Table 11.3 Common options

Option	Default
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

Multiplexing in PPP

- Authentication protocols.
 - Password Authentication Protocol (PAP-0xC023)
 - Challenge Handshake Authentication Protocol (CHAP-protocol value -0xC223)

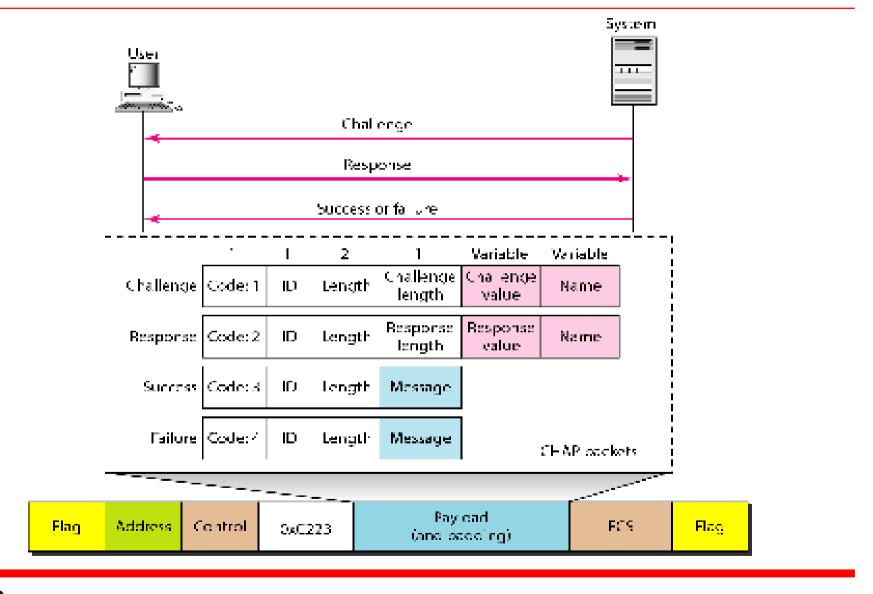
Figure 11.36 PAP packets encapsulated in a PPP frame



Multiplexing in PPP

- CHAP the user password is kept secrete how?
 - The system sends the user a challenge containing a challenge value.
 - The user applies a predefined function to the values to obtain result and sends to the system
 - System does the same and the results are compared

Figure 11.37 CHAP packets encapsulated in a PPP frame



Multiplexing in PPP

Network control Protocol (Protocol value 8021)

One NCP protocol is the Internet Protocol Control

Protocol used to configure the links to carry IP packets.

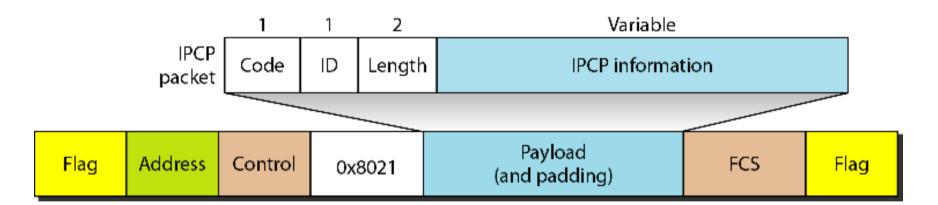
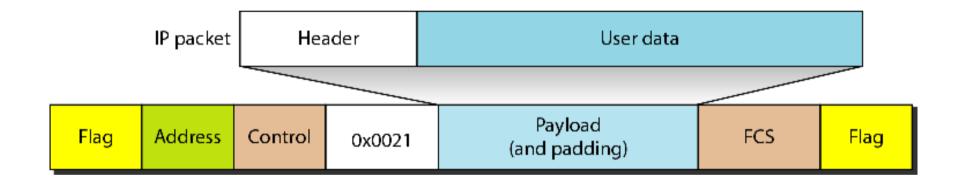


Figure 11.38 IPCP packet encapsulated in PPP frame

Table 11.4 Code value for IPCP packets

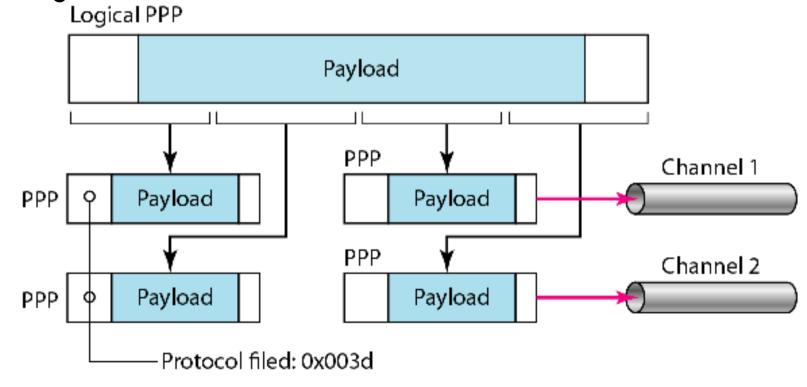
Code	IPCP Packet		
0x01	Configure-request		
0x02	Configure-ack		
0x03	Configure-nak		
0x04	Configure-reject		
0x05	Terminate-request		
0x06	Terminate-ack		
0x07	Code-reject		

Figure 11.39 IP datagram encapsulated in a PPP frame



Multilink PPP

- The availability of multiple channels in a single point to- point link motivated the development
- ·A logical PPP frame is divided into several actual frame





Let us go through the phases followed by a network layer packet as it is transmitted through a PPP connection. Figure 11.41 shows the steps. For simplicity, we assume unidirectional movement of data from the user site to the system site (such as sending an e-mail through an ISP).

The first two frames show link establishment. We have chosen two options (not shown in the figure): using PAP for authentication and suppressing the address control fields. Frames 3 and 4 are for authentication. Frames 5 and 6 establish the network layer connection using IPCP.



Example 11.12 (continued)

The next several frames show that some IP packets are encapsulated in the PPP frame. The system (receiver) may have been running several network layer protocols, but it knows that the incoming data must be delivered to the IP protocol because the NCP protocol used before the data transfer was IPCP.

After data transfer, the user then terminates the data link connection, which is acknowledged by the system. Of course the user or the system could have chosen to terminate the network layer IPCP and keep the data link layer running if it wanted to run another NCP protocol.

Figure 11.41 An example

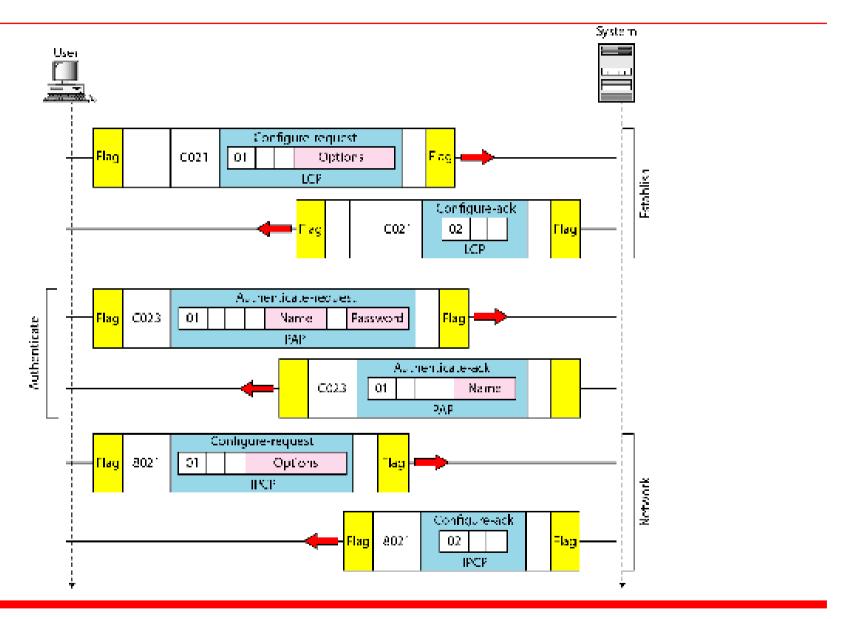


Figure 11.41 An example (continued)

